# File descriptions

|  |  |
| --- | --- |
| 5state.html | UI prototype |
| Recombinase-Based Circuit Design.pdf | Doc I made describing the project for a statistics class. Read this for background info and a description of the CFG I designed for arbitrary-size circuits |
| Gates.jpg | Layout of all two-input-one-output gates, from Siuti, Yazbek & Lu 2013 |
| circuitEnumerator.py | Script to enumerate all valid circuits up to a given number of parts |
| circuitEnumerator2.py | Attempt to recreate the above by using depth-first search on a CFG |
| circuitEnumerator\_14pieces.py | Script to enumerate all valid circuits in the 5-state architecture represented in the UI |
| Table plan.xlsx | Proposed architecture to hold results to be queried by the UI |
| DBnotes.docx | Definition of terms and thoughts about the DB |

# Background

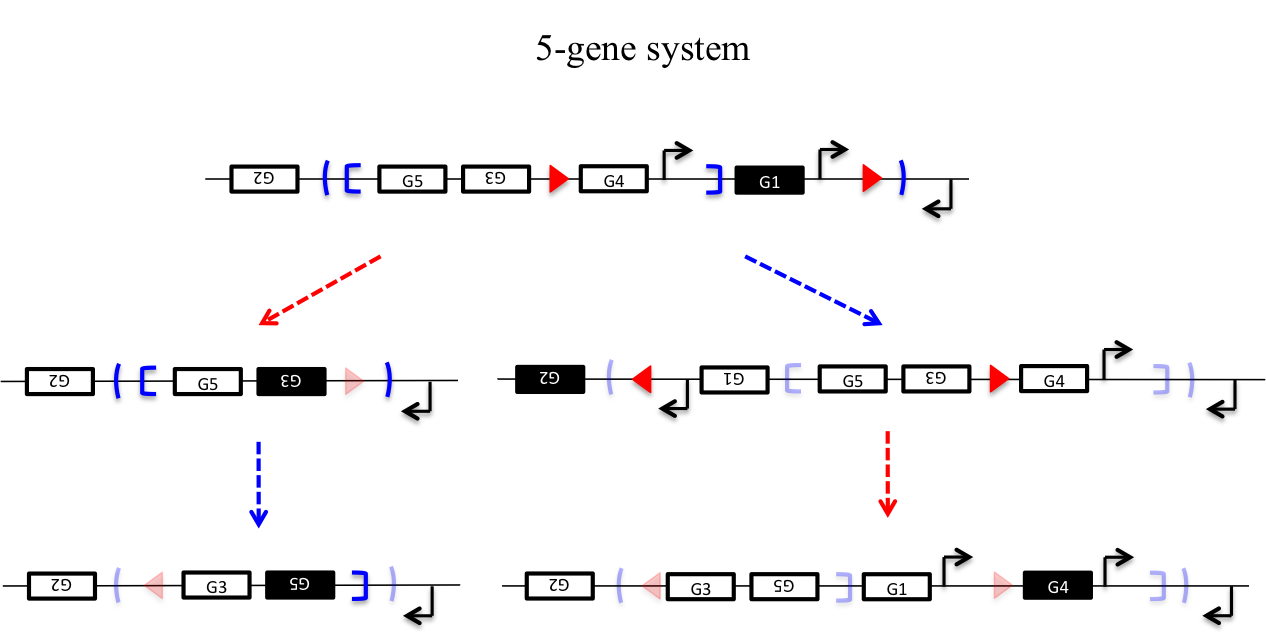
Recombinases: If two recombinases flanking a gene are in opposite orientation when they're activated, it will invert. If they have the same orientation, it will excise that region instead. Once the flip happens, the recombination sites become inactive so they're not going to flip back.

Terminators: Stop transcription, but not perfectly. Some of them only terminate in one direction, but the best tend to be symmetric. We're trying to avoid using unidirectional terminators for that reason.

Circuits are already built (by hand) for all 16 combinational dual-input logic gates (A not B, A and B…)

We want to consider larger truth tables (starting with 3X3) and also create circuits where the order of inputs matters.

5 state machines are achieved by using pairs of orthogonal recombination sites that recognize the same recombinase. The sites don't cross-pollenate, but they can be nested/overlapped to create a much larger state space.



Note that this architecture is order dependent: you get a different result from activating red-blue or blue-red.

# Project Description

Short term goal: some sort of tool where you can specify the output for each of the five states, and it tells you what to make.

GUI for creating states, which then looks up results in a table with all the outputs (created by brute force for now) and tells you what circuit to build.

Output of the GUI should include a Pigeon image similar to the above, and an SBOL-like description.

# Project Status

See “Recombinase-Based Circuit Design.pdf” for a deeper discussion of the project goals and theory.

Stage 1 should be finishing the 5state.html UI. This requires:

* Enumerate legal circuits, and store in a database the UI can query
* Implement a scoring system to decide between circuits with the same truth table (store this score as well)
* Query the DB based on user input, return the best circuit and transform into the desired output format.

The UI is used by dragging “gene” boxes into the corresponding cell state node.

Longer-term goals could include fixing the CFG, then implementing graph search to create circuits of arbitrary size or connectivity.

# Design Considerations

Rules for optimal solutions:

* Avoid multiple copies of the same gene
* Avoid unidirectional terminators that require readthrough
* Minimize number of parts
* Avoid gene expression that involves reading through a promoter in the opposite direction (polymerase binding on the reverse strand will reduce efficiency)

Possible outputs for states: Disabled/Inaccessible/Not Used (not applicable to root), Ambiguous (I specifically don't care), Outputs 1-n. Make the user specifically state as much as they can.

Multiple genes can be expressed in one state.

The 5-state circuit is not symmetrical- this means the mirror image of a user input graph (transform 2<->3, 4<->5) will be a completely different device and may turn out to be easier to create. If the spec is asymmetrical, return two separate lists of devices- one for the original graph, and one for the mirrored graph.

The limiting factor is how many recombinases are available: that will tell us how many states we really can get.